There are no degrees of completeness to these facets of development. These competencies are atomic in nature in which they are wholly true, or they are not. This assessment is not a measure of performance or success, but a tool for attaining a baseline by and for development teams.

Upon completion of the assessment the self-organizing team(s) should leverage the results to formulate a plan forward to improve upon their current state.

|  |  |  |
| --- | --- | --- |
| **Continuous Integration** | | |
| Level 1 - Initial | | |
|  | Developers have long lived branches and perform branch syncing to get the most recent merged changes |  |
|  | Developer branch commits include relatively large numbers of changes |  |
|  | Complex manual workflows are needed to manage code merges (e.g. Git-Flow, code review gates, manual testing) |  |
|  | Pre-production environments are often unavailable and divert resources from development efforts |  |
|  | Build failures involve manual processes and effort to resolve |  |
|  | Build failure alerts do not exist or have a limited audience |  |
| Level 2 - Managed | | |
|  | Every developer commits at least daily to trunk |  |
|  | Every commit triggers an automated build |  |
|  | Every commit triggers automated tests |  |
|  | When the build fails it is fixed within 10 minutes |  |
|  | When any test fails it is fixed within 10 minutes |  |
| Level 3 - Defined | | |
|  | No branch lives past end of day |  |
|  | Every developer commits to trunk multiple times per day |  |
|  | Every developer follows test first development practices (e.g. TDD, BDD, ATDD) |  |
|  | Branch changes have been peer reviewed prior to commit |  |
|  | Branch changes have been tested locally prior to commit |  |
|  | Branch changes have been static scanned locally prior to commit |  |
|  | Developers have local VMs/Containers to deploy and test against prior to merge |  |
| Level 4 - Optimized | | |
|  | Local developer VMs/Containers have been standardized, automated and published to accelerate development and onboarding of new team members (e.g. Docker Compose / Docker Hub) |  |
|  | Pre-prod environments are monitored and automated alerts inform the entire team when there is a problem |  |
|  | SCM and all environments are enabled with automated rollback controls via CI orchestrator |  |

|  |  |  |
| --- | --- | --- |
| **Continuous Delivery/Deployment** | | |
| Level 1 – Initial | | |
|  | Release frequency is sporadic and site reliability is low |  |
|  | Pre-production environments are leveraged providing purpose-built spaces for development processes and internal dev partners |  |
|  | Version control tooling is non-existent or causes undue effort upon the development teams |  |
|  | Deployment to all environments is a largely manual process |  |
|  | Environments are provisioned manually by a centralized infrastructure organization and are bare metal or virtual |  |
|  | Configuration is managed and implemented manually for each independent environment |  |
|  | Data migrations are often required to introduce functional changes and performed manually |  |
| Level 2 – Managed | | |
|  | Release frequency has an establish cadence but often slips but site reliability is stable |  |
|  | Most, if not all, pre-production environments have been replaced with IaaS point of need virtual environments |  |
|  | Version control is utilized to manage source code artifacts |  |
|  | Deployment to all environments is mostly automated but requires human intervention for progression (pipeline not fully automated, validation tollgates, etc.) |  |
|  | Environment provisioning is largely manual and proprietary, but certain technology stacks have some automated provisioning capabilities available |  |
|  | Configuration has been externalized and versioned |  |
|  | Data changes are automated and versioned with the application |  |
| Level 3 – Defined | | |
|  | Releases are consistently frequent and stable |  |
|  | Blue/Green environment strategy has been implemented and final testing takes place in the idle production segment |  |
|  | Version control is used to manage everything necessary to production delivery (code, configuration, scripts, etc.) |  |
|  | Deployment to pre-production environments is fully automated including gating, validation and rollback |  |
|  | Configuration management is orchestrated through automation tools (e.g. chef, puppet, etc.) |  |
|  | The same deployment process is used for all environments |  |
|  | Primary workflow synthetic traffic is generated and monitored and appropriate alerting triggers team action |  |
|  | Continuous Deployment has been established as fully automated build and delivery pipelines push artifacts to production without human interaction. |  |
|  | Data systems have been migrated to new solutions allowing for asset versioning and decoupling from the application |  |
| Level 4 - Optimized | | |
|  | Releases are a non-event and happen continuously multiple times per day |  |
|  | No manual tollgates prevent artifacts from automatically deploying to production and self-healing systems are in place to ensure customer perception of quality is not impacted |  |
|  | Environment provisioning and teardown is fully automated and point-of-need as part of the CICD pipeline |  |
|  | Every developer commit is tested, built, deployed, tempered and enabled in production with full automation continuously with no downtime or negative customer impact |  |
|  | Automated tempering traffic redirection sends a minimally responsible level of user traffic to new environments as they come online |  |
|  | APM monitoring and automated rollback gracefully direct traffic back to last known good revisions when fault tolerances are violated creating self-healing systems with minimized blast radius |  |
|  | Focus has shifted from data issues and migrations to performance and optimization |  |

|  |  |  |
| --- | --- | --- |
| **Security** | | |
| Level 1 – Initial | | |
|  | Security is largely validated by a centralized external group through security review pull requests made by the teams |  |
|  | Security professionals or teams are engaged as a requirement of production release |  |
|  | Transparency into security vulnerabilities is reserved to centralized security or a subset of the team |  |
| Level 2 – Managed | | |
|  | Security is engaged at the earliest stages of ideation and design to assist teams in defining appropriate countermeasures and architectures |  |
|  | Security development tools such as static & dynamic code analysis are leveraged by developer teams to scan on their local development environments |  |
|  | Security scanning is conducted by a centralized team(s) on a regular basis against production |  |
|  | Transparency and awareness into security vulnerabilities is universal amongst the team |  |
| Level 3 – Defined | | |
|  | Development team members have broad if not deep understanding of security practices and proactively use this knowledge to effectively design and implement appropriate security measures |  |
|  | Static and Dynamic security scans are triggered for every commit as part of the CICD pipeline |  |
|  | Security vulnerabilities are made visible in the product backlog and prioritized for resolution with appropriate urgency |  |
| Level 4 – Optimized | | |
|  | Static and dynamic scanning is fully automated as part of the development process and CICD pipeline |  |
|  | Detected vulnerabilities are fixed during development as standard practice |  |
|  | Vulnerabilities detected in existing code are weighted via standard risk assessment scales (e.g. CVSS, OWASP) and prioritized for appropriate remediation against established SLAs |  |

|  |  |  |
| --- | --- | --- |
| **Test Automation** | | |
| Level 1 – Initial | | |
|  | Testing is in large part manually executed |  |
|  | Test automation is primarily targeted at the UI tier |  |
|  | Test automation is managed as a separate or sub-set of the product backlog |  |
|  | Test automation is triggered manually on request as a latter development phase |  |
|  | Coverage metrics may be utilized but not contextualized with outcome metrics providing broader meaning |  |
|  | Cross team/program coordination is needed to engage development teams in end-to-end (E2E) testing efforts |  |
|  | Test data is created ad hoc and the state of which is dependent on test automation execution |  |
| Level 2 – Managed | | |
|  | Manual testing is minimized to exploratory efforts |  |
|  | Shift-left practices are being employed to accelerate development and improve early detection of defects |  |
|  | Unit test coverage is a part of DoD for all new work and implemented without exception preventing additional technical debt accumulation |  |
|  | Test automation is focused on functionality, follows single responsibility principals and is deterministic in approach |  |
|  | Test automation is leveraged to provide confidence and as a gating mechanism for CICD pipelines |  |
|  | Shift-left and isolation testing has reduced the effort and coordination required for E2E testing |  |
|  | Test data is copied from production and scrubbed of confidential values to provide representational sample sets |  |
| Level 3 – Defined | | |
|  | Manual testing has been further minimized by leveraging risk-based testing strategies and targeting key customer workflows |  |
|  | Test first development practices are standard operating procedure |  |
|  | Gaps in unit test coverage are corrected each Sprint as part of an intentional and orchestrated effort |  |
|  | Test automation is a science and approached from multiple facets expanding past deterministic functional tests to stochastic approaches and nonfunctional aspects of the SUT |  |
|  | Consumers and producers alike have implemented self-service test automation eliminating the need for coordinated testing efforts |  |
|  | Test data is purpose-built with a minimal responsible volume, tooling for ease of customization and teardown to provide a known good state |  |
| Level 4 – Optimized | | |
|  | Test automation has been decoupled from the build process further accelerating the build and deploy pipeline |  |
|  | Test automation is laser focused and execution is sliced around that area of focus allowing for speed of parallelization and enable quick targeting of root cause |  |
|  | SUTs have been replatformed, rearchitected or replaced to decouple dependencies removing the need for extensive E2E efforts |  |
|  | Test automation is an accelerant to the development process giving developers and stakeholders the confidence that defect escape is a rare occurrence and the blast radius is well managed when the rare but inevitable leakage happens. |  |
|  | Test data creation is fully automated and fully representational via tooling eliminating the overhead required to prepare and manicure golden copies |  |